ErUM-Data Wuppertal Update

T. Harenberg, M. Sandhoff, M. Vogel, C. Zeitnitz

Containerization of services: conditions data

- The Wuppertal group is currently active in topic area A: work-package 1 (Containerization of user jobs, Containerization of services)
- Work with **standalone containers** proved seminal to the containerization of ATLAS production workflows
- These images are provisioned with a **local conditions database** (SQLite), a resource typically accessed through the network
- Images are being used at selected HPCs (NERSC) to run detector simulation. Work is starting on deploying images for MC reconstruction and for the dataset formats that derive from it
- ATLAS and CMS are currently are converging on a similar data model for their detector conditions databases (CREST, a RESTful client-server architecture)
- This opens the possibility for developing common tools for handling conditions

Log analysis framework for user jobs

- We are also active in topic area A: Work-package 3 (Workflow Control)
- Drawing on the example provided by the ATLAS job validation and reporting software, we are developing a ***log analysis framework for deployment in containers***
  - **Deployment**: in pre-existing containers where payload execution produces log files. Available via a Git-Repository (Dockerfiles and configuration files for installing the framework as additional container layers)
  - **Core components**:
    - Message collecting and processing unit (structuring of log lines by matching regular expressions)
    - Storage unit (optionally a search-engine index)
    - Software layer for configuration, customization and report generation (JSON formatted summary of errors and anomalous messages)
- **Message processing:** structuring of log lines by matching regular expressions provided by the user. Structured data is stored in a local search-engine index

Framework's core components

- After a review of several well-known open-source projects, we chose fluentd and elasticsearch as the core component services
- **fluentd**: light weight, open-source data collection software project written in Ruby
  - It processes unstructured data from different sources via their corresponding input plugins
  - The data is matched and structured (and possibly filtered and/or enriched) before it is saved to different output formats via their corresponding output plugins
- **elasticsearch**: search engine based on the Lucene library. An elasticsearch output plugin stores the structured data in a local elasticsearch instance via a RESTful interface

- **Software layer**:
  - Initialization scripts for configuring the overall deployment: elasticsearch or file storage, user defined regular expressions and modules
  - A python package with classes and functions, and a main script for steering and customizing data processing
- **The main script** instantiates classes and implements queries to the elasticsearch engine via class functions
- **Functions** to process the structured data returned by queries based on the values of the index fields (e.g. logging severity level)
- **User defined modules** to process unstructured messages. A minimal set is provided to process common errors, e.g., std::bad_alloc
- **A log report** is produced in JSON format containing the full result of the query and identified special messages
- Base classes and minimal implementations will be provided, which can be easily customized

A first prototype as proof of concept

- The first use case is a prototype tuned to analyze ATLAS reconstruction job logs
  - Regular expressions match: ***service***, ***level*** and ***message*** fields
    - The ***level*** field is a discreet ordered record with values: DEBUG, INFO, WARNING, ERROR, CRITICAL, FATAL
  - A JSON job report is generated with all messages with level >= ERROR
  - The report also contains:
    - the **worst message**, or the first message with the highest logging level severity
    - the **first error**, or the first message with a chosen logging level severity

Next steps

- **Immediate steps**
  - Submit the first prototype container to the Grid and generate a job report from the job's log file
  - Test that all components work in a containerized environment
- **To-do's** (components still missing for a first full product release)
  - Modules to process and report on common errors. Provide documentation (or interface) on how users can add their own
  - **Interfaces to structure data and implement queries in ES DSL**
    - Users can provide regular expressions for structuring the data (configures fluentd)
    - Users can provide a list of specific values to query in order to produce a report (e.g., all messages with level = WARNING, ERROR and worse)
  - **Make the job report customizable**. Bases classes are provided. Perhaps documentation is enough
  - **Support other report formats**: text, XML and pickle (currently JSON)
  - **Address performance issues**, compare different solutions for core components (e.g., Lucene vs ES)